A decision procedure for term algebras with queues

Tatiana Rybina[*] Andrei Voronkov[†]

Department of Computer Science
University of Manchester
{rybina|voronkov}@cs.man.ac.uk

Abstract

In software verification it is often required to prove statements about heterogeneous domains containing elements of various sorts, such as counters, stacks, lists, trees and queues. Any domain with counters, stacks, lists, and trees (but not queues) can be easily seen a special case of the term algebra, and hence a decision procedure for term algebras can be applied to decide the first-order theory of such a domain.

We present a quantifier-elimination procedure for the first-order theory of term algebras extended with queues. The complete axiomatization and decidability of this theory can be immediately derived from the procedure.

1 Introduction

Our motivation for investigating decidability of term algebras with queues comes from software and hardware verification. In framework of Rainbow project [see Visser, Barringer, Fellows, Gough and Williams 1998], we had to prove correctness of some protocols for infinite-state systems, containing a queue of messages. The correctness problem was reduced to proving first-order statements about such queues. When we replaced the queue by a stack, we could use standard decision procedures for term algebras, since stacks can be easily interpreted in a many-sorted version of term algebras. However, such a replacement changes the protocol and does not prove correctness of the original protocol. When we looked into computer science and logical literature, to our surprise we could not find any decision procedure for the first-order theory of queues. We have found that queues have attracted a lot of attention in verification [Bjørner 1998, Bjørner 2000]. However, [Bjørner 1998, Bjørner 2000] presented a decision procedure only for the existential theory of queues (though in a much richer signature than ours), which was not enough for our purpose: some statements, like absence of deadlocks required truly first-order formulas with quantifier alternations.

In this paper we prove decidability of a first-order theory of term algebras extended with queues. This result implies decidability of the first-order theory of essentially any domain that may contain the following kinds of objects: queues, lists, trees, stacks, finite domains, counters, or embedded domains, like queues of trees or queues of queues, since all these kinds of objects except for queues can be represented as sorts in a many-sorted term algebra.

Independently, our results are of general interest in logic since they give quantifier elimination and decision procedures for the first-order theory of words on any finite alphabet with two operations: addition of alphabet symbols to the left and to the right of a word.

Our proof is based on a quantifier elimination procedure that extends standard quantifier elimination procedures for term algebras [Maĺcev 1961, Maher 1988, Comon and Lescanne 1989, Kunen 1987, Belegradek 1988*a*, Hodges 1993]. More precisely, we use the technique of decomposition functions introduced independently by Kunen [1987] and by Belegradek [1988*a*], and explained in [Hodges 1993][1]. Our quantifier elimination for queues is based on handling quantifier-free formulas expressing periodicity of queues. Such formulas also play a major role in [Bjørner 2000].

For verification purposes, quantifier elimination is even more attractive than a mere decidability result, since quantifier elimination allows one to "simplify" formulas into equivalent quantifier-free formulas (though the size of the quantifier-free formula may be huge). So, one can "reuse" the same subformulas occurring in verification statements by running the quantifier elimination procedure on each subformula once, instead of re-running a decision proce-

[*]Partially supported by a grant from EPSRC (Rainbow project).

[†]Partially supported by grants from EPSRC and the Faculty of Science and Engineering.

[1]There is a inaccuracy in the inductive argument of [Hodges 1993], however it is easy to fix, see the full version of this paper [Rybina and Voronkov 2000].

dure from scratch.

Our paper is structured as follows. In Section 2 we define the structure $\mathcal{Q}(\mathcal{S})$ of term algebras with queues. Section 3 is devoted to the proof of quantifier elimination for the first-order theory of $\mathcal{Q}(\mathcal{S})$. The quantifier elimination procedure and the proof of this section provides an axiomatization for term algebras with queues, this axiomatization is presented in Section 4. In Section 5 we discuss related papers, mostly on term algebras, and some possible results that can be proved for the first-order theories of queues. Some proofs are omitted, they can be found in the full version of this paper [Rybina and Voronkov 2000].

2 Term algebras with queues

2.1 Sorts

We assume a finite set of *basic sorts*. We define *sorts* as follows: (i) every basic sort is a sort; (ii) if α is a sort, then $queue(\alpha)$ is a sort, called a *queue sort*.

We call a *function type* any expression of the form $\alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$, where $\alpha_1, \ldots, \alpha_n, \beta$ are sorts, $n \geq 0$. When $n = 0$, we will write β instead of $\rightarrow \beta$. A *signature* $\mathcal{S}$ is a pair consisting of a finite set of function symbols, and a function : mapping every function symbol f to a function type $\alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$, called the *type of* f, where β is a basic sort. The number n is called the *arity* of f. Function symbols of arity 0 are called *constants*. *We assume that every signature contains at least one constant.*

Though we deal with algebras, i.e., structures with no relation symbols besides the equality $=$, we will sometimes need to extend them by relation symbols of different types. We call a *relation type* any expression of the form $\alpha_1 \times \ldots \times \alpha_n$, where $\alpha_1, \ldots, \alpha_n$ are sorts, $n \geq 0$.

2.2 Structure $\mathcal{Q}(\mathcal{S})$

Every signature $\mathcal{S}$ defines a unique structure $\mathcal{Q}(\mathcal{S})$, called a *term algebra with queues* as follows.

Universe of $\mathcal{Q}(\mathcal{S})$. The *universe of* $\mathcal{Q}(\mathcal{S})$ is defined inductively as follows.

1. For every constant $a : \beta$, a is an element of $\mathcal{Q}(\mathcal{S})$ of the sort β.
2. For every elements $e_1, \ldots, e_n$ of the same sort α, where $n \geq 0$, the sequence $e_1 \ldots e_n$ is an element of $\mathcal{Q}(\mathcal{S})$ of the sort $queue(\alpha)$. Elements of any sort $queue(\alpha)$ will be called *queues*. The empty sequence is called the *empty queue*, and denoted ε_α. Note that there exist empty queues of different sorts. Alternatively, we can characterize the domain of $queue(\alpha)$ as the set of words on α.
3. If $f : \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$ and $e_1, \ldots, e_n$ are elements of $\mathcal{Q}(\mathcal{S})$ of the sorts $\alpha_1, \ldots, \alpha_n$, respectively, then $f(e_1, \ldots, e_n)$ is an element of $\mathcal{Q}(\mathcal{S})$ of the sort β.

- Sort (Natural numbers) :
  1. 0 : nat;
  2. successor : nat $\rightarrow$ nat
- Sort (Negative numbers) :
  1. -1 : neg;
  2. predecessor : neg $\rightarrow$ neg
- Sort (Integers) :
  1. pos : nat $\rightarrow$ int;
  2. neg : neg $\rightarrow$ int
- Sort (Stacks):
  1. empty : stack;
  2. push : int, stack $\rightarrow$ stack

$$\begin{aligned}\textbf{next}(x, y) \quad \Leftrightarrow \quad & \forall z\, (x = \textbf{pos}(z) \rightarrow \\ & \quad y = \textbf{pos}(\textbf{successor}(z))) \wedge \\ & (x = \textbf{neg}(-1) \rightarrow y = \textbf{pos}(0)) \wedge \\ & \forall z\, (x = \textbf{neg}(\textbf{predecessor}(z)) \rightarrow \\ & \quad y = \textbf{neg}(z))\end{aligned}$$

Figure 1: Example: defining several domains and the integer successor function in term algebras

We consider two elements equal only if they have the same sort and coincide as expressions. For example, $f(a) \neq f(b)$ if a and b are different constants. Two queues $e_1 \ldots e_n$ and $d_1 \ldots d_m$ are equal if and only if $n = m$ and $e_1 = d_1, \ldots, e_n = d_n$. The *length* of a queue $q = e_1 \ldots e_n$ is a number of elements of q, i.e. n.

To give the reader the idea of how different domains can be specified in term algebras with sorts, in Figure 1 we define several sorts, including integers and stacks of integers, and write down a first-order formula **next** expressing the successor function on integers.

Using different signatures we obtain different domains, in which sorts can be empty or nonempty, finite or infinite. We are not interested in having empty sorts (and all the resulting complications of first-order logic with function symbols and empty sorts), so we assume that the signature $\mathcal{S}$ is such that every sort defined in it is nonempty. Not that the emptyness of a sort can be effectively decided.

Language of $\mathcal{Q}(\mathcal{S})$. The *language of* $\mathcal{Q}(\mathcal{S})$, denoted $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$, is a first-order language using, in addition to function symbols in $\mathcal{S}$, the equality relation symbol $=$ and the following symbols:

1. for every sort α, the constant $\varepsilon_\alpha : queue(\alpha)$;

2. for every sort α, the function symbols $ladd_\alpha : \alpha \times queue(\alpha) \to queue(\alpha)$ and $radd_\alpha : \alpha \times queue(\alpha) \to queue(\alpha)$.

As usual, we assume an infinite set of variables of each sort α. Intuitively, $ladd$ and $radd$ are the left- and right-insert operations on queues respectively. We denote by $\top$ and $\bot$ the logical connectives "true" and "false".

Semantics of $\mathcal{Q}(\mathcal{S})$. Now we define the interpretation for each symbol of the language $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$.

For every function symbol $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$ of $\mathcal{S}$, the interpretation $\underline{f}$ of this symbol in $\mathcal{Q}(\mathcal{S})$ is defined as follows: for all elements $e_1, \ldots, e_n$ of the sorts $\alpha_1, \ldots, \alpha_n$ we have $\underline{f}(e_1, \ldots, e_n) = f(e_1, \ldots, e_n)$.

Likewise, the interpretation $\underline{\varepsilon_\alpha}$ of the constant ε_α is the empty queue of the sort $queue(\alpha)$, i.e., ε_α itself.

The interpretations $\underline{ladd_\alpha}$ and $\underline{radd_\alpha}$ of the function symbols $ladd_\alpha$ and $radd_\alpha$ are defined as follows. For all elements $e, e_1, \ldots, e_n$ of a sort α:

$$\begin{aligned} \underline{ladd_\alpha}(e, e_1 \ldots e_n) &= ee_1 \ldots e_n; \\ \underline{radd_\alpha}(e, e_1 \ldots e_n) &= e_1 \ldots e_n e. \end{aligned}$$

In computer science literature, queues are usually defined as datatypes with three operations $push$, top and pop, where $push$ inserts an element into the queue from the right, top gives the leftmost element of the queue and pop removes the leftmost element. Using queues with the operations $ladd$ and $radd$ instead gives us a definitionally equivalent first order language as we will now show. We write down explicit definitions of $push_\alpha$, top_α and pop_α in terms of $ladd_\alpha$ and $radd_\alpha$ and vice versa.

$$\begin{aligned} push_\alpha(x, y) = z &\leftrightarrow radd_\alpha(x, y) = z; \\ top_\alpha(x) = y &\leftrightarrow \exists z\, ladd_\alpha(y, z) = x; \\ pop_\alpha(x) = y &\leftrightarrow \exists z\, ladd_\alpha(z, y) = x. \end{aligned}$$

The operations $ladd_\alpha$ and $radd_\alpha$ can be defined in terms of $push_\alpha$, top_α and pop_α as follows:

$$\begin{aligned} & ladd_\alpha(x, y) = z \leftrightarrow \\ & \qquad z \neq \varepsilon_\alpha \wedge top_\alpha(z) = x \wedge pop_\alpha(z) = y; \\ & radd_\alpha(x, y) = z \leftrightarrow push_\alpha(x, y) = z. \end{aligned}$$

In term algebras *without queues*, the interpretation $\underline{t}$ of every ground term t is t itself. Since the domain consists of the ground terms of $\mathcal{S}$, every element of the domain is the interpretation of a unique ground term. In term algebras *with queues*, every element of the domain is also the interpretation of some ground term, but this term is not unique. For example, the queue ab consisting of two constants a and b of a sort α is the interpretation of four different ground terms:

$$\begin{aligned} & ladd_\alpha(a, ladd_\alpha(b, \varepsilon_\alpha)); \\ & ladd_\alpha(a, radd_\alpha(b, \varepsilon_\alpha)); \\ & radd_\alpha(b, ladd_\alpha(a, \varepsilon_\alpha)); \\ & radd_\alpha(b, radd_\alpha(a, \varepsilon_\alpha)). \end{aligned}$$

LEMMA 2.1 *Each element of $\mathcal{Q}(\mathcal{S})$ is the interpretation of a ground term. For every element e of $\mathcal{Q}(\mathcal{S})$ there exists a unique ground term t such that e is the interpretation of t and t has no occurrences of $radd_\alpha$. There exists an algorithm that, given two ground terms s, t, verifies whether $\mathcal{Q}(\mathcal{S}) \models s = t$.* $\square$

Due to this lemma, we can (and will) identify elements of the universe of $\mathcal{Q}(\mathcal{S})$ with ground terms, and thus blur the distinction between syntax and semantics. In this paper we understand validity and equivalence as validity and equivalence in $\mathcal{Q}(\mathcal{S})$.

3 Quantifier elimination

We say that a theory T *admits quantifier elimination* if for any formula $A(\bar{v})$ of the language of this theory, one can effectively find a quantifier-free formula $B(\bar{v})$ of the same language equivalent to $A(\bar{v})$ in T.

The first-order theory of term algebras in the language $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$ does not admit quantifier elimination. A simple example is the formula $A(x) = \exists y(x = f(y))$ expressing that the outermost function symbol of x is f. This formula is not equivalent to any quantifier-free formula of the language $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$. However, we can change $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$ into a definitionally equivalent language $\mathcal{L}'_{\mathcal{Q}(\mathcal{S})}$ which admits quantifier elimination.

We introduce the following function and relation symbols:

1. a unary relation symbol $\mathbf{Is}_f$ for each function symbol $f : \alpha_1 \times \ldots \times \alpha_n \to \beta, n \geq 1$ of $\mathcal{S}$, this symbol is of the type β;

2. a unary function symbol f_i, called a *decomposition function*, for each function symbol $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$, $n \geq 1$ of $\mathcal{S}$ and each $i = 1, \ldots, n$, this symbol is of the type $\beta \to \alpha_i$;

3. unary function symbols

$$\begin{aligned} & lhead_\alpha : queue(\alpha) \to \alpha, \\ & rhead_\alpha : queue(\alpha) \to \alpha, \\ & ltail_\alpha : queue(\alpha) \to queue(\alpha), \\ & rtail_\alpha : queue(\alpha) \to queue(\alpha) \end{aligned}$$

for each sort α. The symbols $lhead$ and $rhead$ are called *left and right head functions*. The symbols $ltail$ and $rtail$ are called *left and right tail functions*.

For convenience, we will usually omit types in function and relation symbols and simply write $ladd, radd, \ldots$ etc. The first-order language using the constants of $\mathcal{S}$ (including ε_α) and these new symbols will be denoted $\mathcal{L}'_{\mathcal{Q}(\mathcal{S})}$. Note that all function symbols of $\mathcal{L}'_{\mathcal{Q}(\mathcal{S})}$ are unary. In our proofs we will sometimes use formulas with symbols both in $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$ and $\mathcal{L}'_{\mathcal{Q}(\mathcal{S})}$.

Now, we define interpretations of the new function and relation symbols. We want to avoid having partial functions, so we will have to give artificial values to some functions on some arguments. To this end, we choose one element in each sort α and declare it the value of these terms. The fixed element will be denoted by fix_α. When α is clear from the context, we simply write fix. For simplicity, we assume that for any queue sort $queue(\alpha)$ the fixed element is the empty queue ε_α.

1. $\mathbf{Is}_f(x)$ denotes that the outermost function symbol of x is f, i.e. $\mathbf{Is}_f(x) \leftrightarrow \exists x_1 \ldots x_n\, f(x_1, \ldots, x_n) = x$.

2. f_i's are decomposition functions, they extract the ith component from the terms of the form $f(t_1, \ldots, t_n)$:

$$\begin{array}{l} f_i(x) = y \leftrightarrow \\ \quad \forall x_1, \ldots, x_n (f(x_1, \ldots, x_n) = x \rightarrow y = x_i) \wedge \\ \quad (\neg \mathbf{Is}_f(x) \rightarrow y = \mathsf{fix}_\alpha). \end{array}$$

   The rest of the new functions are intended to decompose queues.

3. The functions $lhead$ and $ltail$ give the leftmost element of a queue, and the queue obtained by removing this element:

$$\begin{array}{l} lhead(q) = y \leftrightarrow \\ \quad \exists w (ladd(y, w) = q) \vee (q = \varepsilon \wedge y = \mathsf{fix}); \\ ltail(q) = w \leftrightarrow \\ \quad \exists y (ladd(y, w) = q) \vee (q = \varepsilon \wedge w = \varepsilon). \end{array}$$

4. Likewise, the functions $rhead$ and $rtail$ give the rightmost element of a queue, and the queue obtained by removing this element.

We will leave out parentheses in successive applications of $lhead, rhead, ltail, rtail$ and the decomposition functions, for example, we write $f_i rhead_\alpha ltail_\alpha x$ instead of $f_i(rhead_\alpha(ltail_\alpha(x)))$. Uppercase letters L, M stand for sequences, maybe empty, of any unary function symbols of $\mathcal{L}'_{\mathcal{Q}(\mathcal{S})}$, we often call such sequences *prefixes*. For example, if $L = rtail_\beta f_i lhead_\alpha$ then Lx denotes the term $rtail_\beta f_i lhead_\alpha x$.

Let us note that the new collection of function symbols is definitionally equivalent to the collection of function symbols of $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$. We have already shown above how to express the symbols of the new language using first-order formulas of the old language, now we will express the old language symbols using the new ones. Indeed, it is not hard to argue that the following formulas are tautologies:

$$\begin{array}{l} ladd(x, y) = z \leftrightarrow \\ \quad z \neq \varepsilon \wedge lhead(z) = x \wedge ltail(z) = y; \\ radd(x, y) = z \leftrightarrow \\ \quad z \neq \varepsilon \wedge rhead(z) = x \wedge rtail(z) = y; \\ f(x_1, \ldots, x_n) = y \leftrightarrow \mathbf{Is}_f(y) \wedge \bigwedge_{i=1 \ldots n} f_i(y) = x_i. \end{array}$$

In [Rybina and Voronkov 2000] we impose some restrictions on the fixed elements of sorts. These conditions guarantee that for every element e and nonempty sequence of unary functions L, if $Le = e$, then e is the fixed element of its sort. We do not give details here, since we omit the proof where this property is used.

Our aim in this section is to prove the following theorem:

THEOREM 3.1 (Quantifier Elimination) *The first-order theory of term algebras with queues in the language $\mathcal{L}'_{\mathcal{Q}(\mathcal{S})}$ admits quantifier elimination.*

The proof of this theorem will be based on a series of transformations of $A(\bar{v})$ into equivalent formulas of special form, until we obtain a quantifier-free formula. Let us now introduce a convenient definition.

DEFINITION 3.2 (quantifier elimination reduction class) Let $\mathfrak{C}_1$ and $\mathfrak{C}_2$ be two classes of formulas. We say $\mathfrak{C}_1$ is a *quantifier elimination reduction class for* $\mathfrak{C}_2$ if we have: every formula $A \in \mathfrak{C}_2$ is equivalent to a quantifier-free formula if so is every formula of $\mathfrak{C}_1$. A class $\mathfrak{C}$ is called an *absolute quantifier elimination reduction class*, if it is a quantifier elimination reduction class for the class of all formulas.

Let us note two main properties of quantifier elimination reduction classes.

1. The first-order theory of term algebras admits quantifier elimination if and only if the class of all quantifier-free formulas of this theory is an absolute quantifier elimination reduction class.

2. If $\mathfrak{C}_1$ is a quantifier elimination reduction class for $\mathfrak{C}_2$ and $\mathfrak{C}_2$ is a quantifier elimination reduction class for $\mathfrak{C}_3$, then $\mathfrak{C}_1$ is a quantifier elimination reduction class for $\mathfrak{C}_3$.

3. If $\mathfrak{C}_1$ is a quantifier elimination reduction class for $\mathfrak{C}_2$ and $\mathfrak{C}_2$ is an absolute quantifier elimination reduction class, then $\mathfrak{C}_1$ is an absolute quantifier elimination reduction class.

Speaking in new terms, we will provide a sequence of absolute quantifier elimination reduction classes.

We call a *literal* any atomic formula or its negation. Consider the class of formulas of the form

$$\exists x(A_1 \wedge \ldots \wedge A_n), \tag{1}$$

where $n > 0$ and all A_i's are literals.

LEMMA 3.3 *The class of formulas (1) is an absolute quantifier elimination reduction class.* □

Now it is enough to show how to eliminate quantifiers from formulas (1). Our proof will be similar to the proof of quantifier elimination for term algebras as presented in [Hodges 1993], but with one essential difference. First-order formulas of the theory of term algebras cannot express very "deep" properties of terms. For example, one cannot express that c is the only constant occurring in a term x. The first-order language of queues can express certain statements about elements in a queue at an arbitrary depth. For example, the following formula expresses that c is the only element in a queue q, i.e., $q = c \ldots c$:

$$ladd(c, q) = radd(c, q). \tag{2}$$

This property of the first-order language of queues will create some technical difficulties. Later, we will introduce a special notation that allows us to express formulas like (2).

Consider literals of the following forms:

$$\begin{array}{llll} x = t, & x = Lx, & Lx = t, & Lx = Mx, \\ x \neq t, & x \neq Lx, & Lx \neq t, & Lx \neq Mx, \\ \mathbf{Is}_f(x), & \mathbf{Is}_f(Lx), & \neg\mathbf{Is}_f(x), & \neg\mathbf{Is}_f(Lx), \end{array} \tag{3}$$

where L, M are *nonempty* and t is a term in which x does not occur.

LEMMA 3.4 *The class of formulas $\exists x(C)$, where C is a conjunction of literals of the form (3), is a quantifier elimination reduction class for the class of formulas of the forms (1). Thus, by Lemma 3.3, this class of formulas is an absolute quantifier elimination reduction class.* □

In our proofs we will often use the following trick. Given a formula $C = \exists x((A_1 \vee \ldots \vee A_n) \wedge B)$, we will consider instead formulas $C_i = \exists x(A_i \wedge B)$. Note that $C \leftrightarrow C_1 \vee \ldots \vee C_n$ but C_i do not contain disjunction. We will refer to this trick as *disjunctive splitting*.

It remains to show how to eliminate the existential quantifier applied to a conjunction of literals of forms given in (3). Let us first get rid of some simple cases in (3), so that only the following kind of literals remain:

$$\begin{array}{llll} Lx = t, & Lx = Mx, & \mathbf{Is}_f(x), & \mathbf{Is}_f(Lx), \\ x \neq t, & Lx \neq t, & Lx \neq Mx, & \end{array} \tag{4}$$

where L, M are *nonempty* and t is a term in which x does not occur.

LEMMA 3.5 *The class of formulas $\exists x(C)$, where C is a conjunction of literals of the form (4), is a quantifier elimination reduction class for the class of formulas $\exists x(D)$, where D is a conjunction of formulas (3). Thus, by Lemma 3.4, this class of formulas is an absolute quantifier elimination reduction class.* □

It remains to show how to eliminate the existential quantifier applied to a conjunction of literals of forms given in (4). Our next step would be to get rid of the prefixes L in terms Lx, after this the proof will be relatively straightforward. In the case of quantifier elimination for term algebras, there is a simple way to get rid of such prefixes, but for term algebras with queues variables x of queue sorts will cause some complications. Another complication is caused by the fact that an application of *lhead* or *rhead* to a term of a queue sort may give a term of a basic sort, and vice versa, the application of an decomposition function f_i to a term of a basic sort may give a term of a queue sort. So we will now split the prefixes L to *segments*, so that within each segment the terms have the same queue sort, or otherwise only basic sorts.

Let L be a sequence of unary functions of the new language $\mathcal{L}'_{\mathcal{Q}(\mathcal{S})}$ and $L = L_1 M L_2$, where L_1, L_2 may be empty. We say that M is a *segment* in L in one of the following cases:

1. M is a sequence of the decomposition functions f_i, the last symbol in L_1 and the first symbol in L_2 are *not* decomposition functions (in this case we can say that M is a subsequence consisting of decomposition functions that can be extended neither to the left nor to the right).

2. M is a sequence consisting of a head function followed by zero or more tail functions and the first symbol of L_2 is not a tail function.

3. M is a sequence consisting of one or more tail functions, the first symbol of L_2 is not a tail function, and L_1 is empty.

Segments are illustrated on the following example:

$$\underbrace{ltail\ rtail}_{\text{segment 1}}\ \underbrace{f_1\ h_2\ f_1}_{\text{segment 2}}\ \underbrace{lhead}_{\text{segment 3}}\ \underbrace{rhead\ rtail\ rtail}_{\text{segment 4}}.$$

Let A be any quantifier-free formula and x a variable. We call the *segment depth* of x in A the maximal number of segments in all prefixes L such that Lx occurs in A. If the segment depth is 0, no function symbol is applied to x in A at all.

Recall that our next aim is to eliminate the existential quantifier $\exists x$ applied to a conjunction of literals of forms given in (4). Our next step would be to show how to decrease the segment depth of x. We begin with a special case of the segment depth ≤ 1 and then note that the proof we have used for this case can be generalized to decrease an arbitrary segment depth. We will consider two separate cases, depending on whether x is of a basic sort or not.

LEMMA 3.6 *Consider the class $\mathfrak{C}$ of formulas of the form*

$$\exists x(A_1(x) \wedge \ldots \wedge A_n(x)), \tag{5}$$

where x is of a basic sort β, each A_i is of the form (4), and the segment depth of x in this formula is at most 1. The class of quantifier-free formulas is a quantifier elimination reduction class for $\mathfrak{C}$. □

We want to prove a statement similar to Lemma 3.6, but for queue sorts.

Consider the set of literals

$$\begin{array}{lll} Lq = t, & Lq = Mq, & q \neq t, \\ Lq \neq t, & Lq \neq Mq, & \mathbf{Is}_f(Lq), \end{array} \tag{6}$$

where L, M are *nonempty* and t is a term in which q does not occur.

LEMMA 3.7 *Consider the class $\mathfrak{C}$ of formulas of the form*

$$\exists q(A_1(q) \wedge \ldots \wedge A_n(q)), \tag{7}$$

where q is of a queue sort $queue(\alpha)$, each A_i is of the form (6), and the segment depth of q in C is at most 1. The class of quantifier-free formulas is a quantifier elimination reduction class for $\mathfrak{C}$.

This lemma requires elaborate reasoning and will be proved after a series of other statements. We will explain the proof informally. The proof is carried out using the case analysis on the length of the queue q. For "short" queues, we will eliminate $\exists q$ immediately, for "sufficiently long" ones we will substitute a "sufficiently long" term for q. To understand what is "sufficiently long", we will introduce the notions of left and right depths of q in a formula. For long enough queues we have to understand how to deal with quantifier-free formulas like (2), which express "deep" properties of queues. As we will see, all these deep properties of queues can be formulated as statements about periodicity of queues. So we will introduce special formulas $*_{m,n}$ expressing periodicity of queues, and prove several properties of these formulas. After having done all this preliminary work, quantifier elimination will be more or less straightforward.

Now we will introduce notation for "short" and "long" queues. For terms $t_1, \ldots, t_n : \alpha$, where $n \geq 0$ we denote by $|t_1 \ldots t_n|$ the term

$$ladd(t_1, ladd(t_2, \ldots ladd(t_n, \varepsilon) \ldots)).$$

Evidently, this term denotes the queue consisting of elements $t_1, \ldots, t_n$. Note that $||$ is ε. Likewise, for terms $t_1, \ldots, t_n, s_1, \ldots, s_m : \alpha$ and $w : queue(\alpha)$ we denote by $|t_1 \ldots t_n \underline{w} s_1 \ldots s_m|$ the term

$$\begin{array}{l} ladd(t_1, ladd(t_2, \ldots ladd(t_n, \\ radd(s_m, radd(s_{m-1}, \ldots radd(s_1, w) \ldots))) \ldots)). \end{array}$$

Evidently, this term denotes the queue obtained from w by adding the sequence $t_1, \ldots, t_n$ on the left and the sequence $s_1, \ldots, s_m$ on the right.

For each positive integer m, we denote by $length(q) < m$ the following formula

$$q = \varepsilon \vee \bigvee_{k=1\ldots m-1} \exists x_1 \ldots \exists x_k (q = |x_1 \ldots x_k|).$$

The following lemma asserts that every queue is either "short" (of length less than $l + r - 1$) or "sufficiently long" (of length at least $l + r$).

LEMMA 3.8 *For every nonnegative integers l, r such that $l + r > 0$ the following formula is a tautology:*

$$\begin{array}{l} (length(q) < l + r) \ \vee \\ \exists z_1 \ldots \exists z_l \exists u_1 \ldots \exists u_r \exists w (q = |z_1 \ldots z_l \underline{w} u_1 \ldots u_r|). \end{array} \tag{8}$$

□

PROOF (of Lemma 3.7). We define the *left (right) depth* of q in (7), as the maximal number of $lhead_\alpha$'s and $ltail_\alpha$'s (respectively, $rhead_\alpha$'s and $rtail_\alpha$'s) in all prefixes L such that Lq appears in (7). For example, the left depth of q in the formula $lhead_\alpha ltail_\alpha q \wedge q \neq w$ is 2 and the right depth is 0.

Let the left depth of q in (7) be l and the right depth be r. Using Lemma 3.8 and disjunctive splitting on formula (8), we can replace (7) by a disjunction of formulas of the following two forms:

$$\exists q \exists x_1 \ldots \exists x_k (\begin{array}{l} q = |x_1 \ldots x_k| \wedge \\ A_1(q) \wedge \ldots \wedge A_n(q) \ ), \end{array} \tag{9}$$

$$\begin{array}{l} \exists q \exists z_1 \ldots \exists z_l \ \exists w \exists u_1 \ldots \exists u_r \ (\\ \qquad q = |z_1 \ldots z_l \underline{w} u_1 \ldots u_r| \wedge \\ \qquad A_1(q) \wedge \ldots \wedge A_n(q) \ ). \end{array} \tag{10}$$

So it remains to show how to eliminate quantifiers in such formulas.

Any formula (9) is equivalent to

$$\exists x_1 \dots \exists x_k (A_1(|x_1 \dots x_k|) \wedge \dots \wedge A_n(|x_1 \dots x_k|)).$$

Recall that every $A_i(q)$ is of forms (6). Simplify all literals $A_i(|x_1 \dots x_k|)$ using the following valid equations:

$$\begin{array}{l} \mathit{lhead}(|y_1 \dots y_m|) = y_1, \\ \mathit{ltail}(|y_1 \dots y_m|) = |y_2 \dots y_m|, \\ \mathit{ltail}(||) = ||, \quad \mathit{lhead}(||) = \mathsf{fix}, \\ \mathit{rhead}(|y_1 \dots y_m|) = y_m, \\ \mathit{rtail}(|y_1 \dots y_m|) = |y_1 \dots y_{m-1}|, \\ \mathit{rtail}(||) = ||, \quad \mathit{rhead}(||) = \mathsf{fix}, \end{array}$$

then we obtain a formula of the form

$$\exists x_1 \dots \exists x_k (B_1 \wedge \dots \wedge B_n),$$

where the B_i's are formulas $p = s$, $p \neq s$, $\mathbf{Is}_f(p)$, $|p_1 \dots p_l| = |s_1 \dots s_u|$ and $|p_1 \dots p_l| \neq |s_1 \dots s_u|$, where all p's and s's are either x_i's or terms in which x_i's do not occur. Formulas $|p_1 \dots p_l| = |s_1 \dots s_u|$ for $l \neq u$ can be replaced by $\perp$, and for $l = u$ by $p_1 = s_1 \wedge \dots \wedge p_l = s_l$, and similar for formulas $|p_1 \dots p_l| \neq |s_1 \dots s_u|$. After this transformation and disjunctive splitting we can assume that all B_i's are formulas $p = s$, $p \neq s$, $\mathbf{Is}_f(p)$, then we can proceed as in Lemma 3.6 to get rid of all quantifiers $\exists x_i$.

Now it remains to show how to eliminate quantifiers in (10). Formula (10) is equivalent to

$$\begin{array}{c} \exists z_1 \dots \exists z_l \exists w \exists u_1 \dots \exists u_r (\\ A_1(|z_1 \dots z_l \underline{w} u_1 \dots u_r|) \\ \wedge \dots \wedge \\ A_n(|z_1 \dots z_l \underline{w} u_1 \dots u_r|)). \end{array}$$

Recall that every $A_i(q)$ is of forms (6). Simplify all literals $A_i(|z_1 \dots z_l \underline{w} u_1 \dots u_r|)$ using the following valid equations:

$$\begin{array}{l} \mathit{lhead}(|y_1 \dots y_m \underline{w} v_1 \dots v_k|) = y_1, \\ \mathit{ltail}(|y_1 \dots y_m \underline{w} v_1 \dots v_k|) = |y_2 \dots y_m \underline{w} v_1 \dots v_k|, \\ \mathit{rhead}(|y_1 \dots y_m \underline{w} v_1 \dots v_k|) = v_k, \\ \mathit{rtail}(|y_1 \dots y_m \underline{w} v_1 \dots v_k|) = |y_1 \dots y_m \underline{w} v_1 \dots v_{k-1}|. \end{array}$$

Note that because of the choice of l and r as the left and the right depth of q in (7), this transformation will remove *all* applications of $\mathit{lhead}, \mathit{rhead}, \mathit{ltail}, \mathit{rtail}$ to w from the formula. Further simplify the obtained formula using the following equivalences:

$$\begin{array}{l} |y_1 \dots y_m \underline{w} v_1 \dots v_k| = |p_1 \dots p_l \underline{w} q_1 \dots q_u| \leftrightarrow \\ \quad y_1 = p_1 \wedge \ |y_2 \dots y_m \underline{w} v_1 \dots v_k| = \\ \quad\quad |p_2 \dots p_l \underline{w} q_1 \dots q_u|, \\ |y_1 \dots y_m \underline{w} v_1 \dots v_k| = |p_1 \dots p_l \underline{w} q_1 \dots q_u| \leftrightarrow \\ \quad v_k = q_u \wedge \ |y_1 \dots y_m \underline{w} v_1 \dots v_{k-1}| = \\ \quad\quad |p_1 \dots p_l \underline{w} q_1 \dots q_{u-1}|, \end{array}$$

and similar for $\neq$ instead of $=$. Using disjunctive splitting, when necessary, we obtain a formula of the form

$$\exists x_1 \dots \exists x_k \exists w (B_1 \wedge \dots \wedge B_n),$$

where the B_i's are formulas of the following forms

$$\begin{array}{ll} x_i = x_j, \quad x_i = t, & \mathbf{Is}_f(x_i), \quad w = t, \\ x_i \neq x_j, \quad x_i \neq t, & \quad\quad w \neq t \\ |y_1 \dots y_m \underline{w}| = |\underline{w} q_1 \dots q_u|, & |y_1 \dots y_m \underline{w} v_1 \dots v_k| = t, \\ |y_1 \dots y_m \underline{w}| \neq |\underline{w} q_1 \dots q_u|, & |y_1 \dots y_m \underline{w} v_1 \dots v_k| \neq t, \end{array}$$

where y_i, v_i, q_i are variables from the set $\{x_1, \dots, x_k\}$, and t are terms not containing any x_i's and w.

Evidently, in the equalities $|y_1 \dots y_m \underline{w}| = |\underline{w} q_1 \dots q_u|$ and $|y_1 \dots y_m \underline{w}| \neq |\underline{w} q_1 \dots q_u|$ we can assume $m = u$, otherwise they can be replaced by $\perp$ and $\top$, respectively. Equalities $|y_1 \dots y_m \underline{w} v_1 \dots v_k| = t$ can be dealt with in the standard way: we replace any equation $\mathit{ladd}(y_i, s) = t$ by $t \neq \varepsilon \wedge \mathit{lhead}(t) = y_i \wedge \mathit{rtail}(t) = s$, and similar for radd and for literals $|y_1 \dots y_m \underline{w} v_1 \dots v_k| \neq t$. Equalities $w = t$ can be dealt with as in Lemma 3.5, by removing them and replacing w by t. So we can assume that all B_i'have one of the following forms

$$\begin{array}{l} x_i = x_j, \quad x_i = t, \quad x_i \neq x_j, \\ x_i \neq t, \quad \mathbf{Is}_f(x_i), \end{array} \tag{11}$$

$$\begin{array}{l} |y_1 \dots y_m \underline{w}| = |\underline{w} q_1 \dots q_m|, \\ |y_1 \dots y_m \underline{w}| \neq |\underline{w} q_1 \dots q_m|, \quad w \neq t. \end{array} \tag{12}$$

In equations (11) w does not occur, and the variables x_j's in these equations are of segment depth 0. They can be dealt with as in Lemma 3.5 after we eliminate $\exists w$, so we will only consider the case when all B_i's have one of the forms in (12).

We will now study properties of formulas $|y_1 \dots y_m \underline{w}| = |\underline{w} q_1 \dots q_m|$, and then return to the proof of Lemma 3.7 again. If $w = t$ appears, we can eliminate the quantifier $\exists w$ by replacing w by t in all B_i's. We will still have several quantifiers $\exists x_i$, but the variables x_i are of segment depth 0. □

For all sorts α and integers m, k such that $m > k \geq 0$, denote by $*_{m,k,\alpha}(q, z_1, \dots, z_m)$ the following formula:

$$|z_1 \dots z_m \underline{w}| = |\underline{w} z_{k+1} \dots z_m z_1 \dots z_k|.$$

As usual, we will omit α and write $*_{m,k}$ instead. One can notice that the formula $*_{m,k}$ express some kind of periodicity properties of w. For example, the formula $*_{4,0}(w, a, b, a, b)$ expresses that w is obtained by repeating the sequence ab zero or more times.

We call a sequence of variables or elements $t_1 \dots t_m$ *periodic* if it has the form

$$\underbrace{s_1 \dots s_n \dots s_1 \dots s_n}_{k \text{ times}}, \tag{13}$$

for some $k \geq 2$. In this case we also write $t_1 \dots t_m = (s_1 \dots s_n)^k$. Any nonperiodic sequence of variables is called a *pattern*. A pattern $s_1, \dots, s_n$ is said to be the *pattern of a sequence* $t_1, \dots, t_m$ of variables, if $t_1, \dots, t_m$ has the form (13), for some $k \geq 1$. For example, the sequences x, y, x, y and z, z, z, z are periodic, their patterns are x, y and z respectively, but the sequence x, y, x is not periodic, and hence its pattern is x, y, x itself.

Let $e_1, \dots, e_n$ be a sequence of elements of $\mathcal{Q}(\mathcal{S})$ and $z_1, \dots, z_n$ be a sequence of variables. We call $e_1, \dots, e_n$ a *shadow* of $z_1, \dots, z_n$ if for every i, j, $e_i = e_j$ if and only if z_i coincides with z_j.

For every pattern $p_1, \dots, p_n$ denote by $\#^{p_1,\dots,p_n}(z_1, \dots, z_n)$ the formula

$$\bigwedge_{i<j, p_i=p_j} z_i = z_j \wedge \bigwedge_{i<j, p_i \neq p_j} z_i \neq z_j$$

For example, we have

$$\begin{aligned}\#^{x,y,y,x}(z_1, z_2, z_3, z_4) \leftrightarrow z_1 = z_4 \wedge z_2 = z_3 \wedge \\ z_1 \neq z_2 \wedge z_1 \neq z_3 \wedge z_2 \neq z_4 \wedge z_3 \neq z_4.\end{aligned}$$

It is easy to see that the property "$e_1, \dots, e_n$ is a shadow of $p_1, \dots, p_n$" can be expressed as $\#^{p_1,\dots,p_n}(e_1, \dots, e_n)$.

Let $t_1, \dots, t_m$ be a sequence of terms or elements and $0 \leq k < m$. We call the *k-rotation* of this sequence the sequence $t_{k+1}, \dots, t_m, t_1, \dots, t_k$. We call a *rotation* of a sequence of terms or elements $t_1, \dots, t_m$ any k-rotation of this sequence, for $0 \leq k < m$. For example all rotations of the sequence x, y, z are x, y, z itself, y, z, x and z, x, y. All rotations of x, y, x are x, y, x itself, y, x, x and x, x, y.

LEMMA 3.9 *Suppose $e_1, \dots, e_m$ and $d_1, \dots, d_m$ are elements, w is a queue, and $|e_1 \dots e_m \underline{w}| = |\underline{w} d_1 \dots d_m|$. Then $d_1, \dots, d_m$ is a rotation of $e_1, \dots, e_m$. Therefore, the following formula is valid:*

$$\begin{aligned}&|z_1 \dots z_m \underline{w}| = |\underline{w} u_1 \dots u_m| \rightarrow \\ &\bigvee_{k=0\dots m-1} (z_1 = u_{k+1} \wedge \dots \wedge z_{m-k} = u_m \wedge \\ &\quad z_{m-k+1} = u_1 \wedge \dots \wedge z_m = u_k \wedge \\ &\quad *_{m,k}(w, z_1, \dots, z_m)).\end{aligned} \tag{14}$$

□

Given an equality of the form $|e_1 \dots e_m \underline{q}| = |\underline{q} e_{k+1} \dots e_m e_1 \dots e_k|$, we now want to prove some statements about the sequence $e_1 \dots e_m$. Let us denote by $(e_1 \dots e_r)^*$ any sequence of the form

$$\underbrace{e_1 \dots e_r \ \dots \ e_1 \dots e_r}_{0 \text{ or more times}}$$

LEMMA 3.10 *Let $p_1, \dots, p_m$ be a pattern, $e_1, \dots, e_m$ a sequence of elements of a sort α, and q a queue of the sort $queue(\alpha)$. Then*

$$\#^{p_1,\dots,p_m}(e_1, \dots, e_m) \wedge *_{m,k}(q, e_1, \dots, e_m)$$

holds if and only if $e_1, \dots, e_m$ is a shadow of $p_1, \dots, p_m$ and

$$q = \underbrace{e_1 \dots e_m \ \dots \ e_1 \dots e_m}_{0 \text{ or more times}} e_1 \dots e_k.$$

□

LEMMA 3.11 *Let $z_1, \dots, z_n$ a periodic sequence of variables and $p_1, \dots, p_m$ be its pattern. Then the following formula is valid:*

$$\begin{aligned}&\#^{p_1,\dots,p_m}(z_1, \dots, z_m) \wedge *_{n,l}(q, z_1, \dots, z_n) \leftrightarrow \\ &\#^{p_1,\dots,p_m}(z_1, \dots, z_m) \wedge *_{m,k}(q, z_1, \dots, z_m),\end{aligned} \tag{15}$$

where $k = (l \bmod m)$. □

LEMMA 3.12 *Let $z_1, \dots, z_m$ and $u_1, \dots, u_m$ be two patterns and $k \neq l$. Then the following formula is valid*

$$\begin{aligned}&\#^{z_1,\dots,z_m}(z_1, \dots, z_m) \wedge \\ &\#^{u_1,\dots,u_m}(u_1, \dots, u_m) \wedge *_{m,k}(q, z_1, \dots, z_m) \\ &\quad \rightarrow \neg *_{m,l}(q, u_1, \dots, u_m).\end{aligned} \tag{16}$$

□

LEMMA 3.13 *Let $z_1, \dots, z_m$ and $u_1, \dots, u_n$ be two patterns and $m \neq n$. Then the following formula is valid*

$$\begin{aligned}&\#^{z_1,\dots,z_m}(z_1, \dots, z_m) \wedge \\ &\#^{u_1,\dots,u_n}(u_1, \dots, u_n) \wedge \\ &*_{m,k}(q, z_1, \dots, z_m) \wedge *_{n,l}(q, u_1, \dots, u_n) \rightarrow \\ &\qquad \mathit{length}(q) < m + n.\end{aligned} \tag{17}$$

□

LEMMA 3.14 *Let $z_1, \dots, z_m$ and $u_1, \dots, u_m$ be two patterns. Then the following formula is valid*

$$\begin{aligned}&\#^{z_1,\dots,z_m}(z_1, \dots, z_m) \wedge \\ &\#^{u_1,\dots,u_m}(u_1, \dots, u_m) \wedge \\ &*_{m,k}(q, z_1, \dots, z_m) \wedge *_{m,k}(q, u_1, \dots, u_m) \rightarrow \\ &q = |z_1, \dots, z_k| \vee (z_1 = u_1 \wedge \dots \wedge z_m = u_m).\end{aligned} \tag{18}$$

□

Now we can conclude the proof of Lemma 3.7.

PROOF (of Lemma 3.7, continued). Recall that we have to show how to eliminate the quantifier in a formula C of the form $\exists w(B_1 \wedge \ldots \wedge B_p)$, where all B_i's have one of the forms given in (12):

$$|y_1 \ldots y_n \underline{w}| = |\underline{w} q_1 \ldots q_n|,$$
$$|y_1 \ldots y_n \underline{w}| \neq |\underline{w} q_1 \ldots q_n|, \quad \underline{w} \neq t.$$

By Lemma 3.9 one can assume that in all such formulas $q_1 \ldots q_n$ is a rotation of $y_1 \ldots y_n$, so all B_i's have one of the forms

$$|y_1 \ldots y_n \underline{w}| = |\underline{w} y_{l+1} \ldots y_n y_1 \ldots y_l|,$$
$$|y_1 \ldots y_n \underline{w}| \neq |\underline{w} y_{l+1} \ldots y_n y_1 \ldots y_l|, \quad w \neq t,$$

i.e. one of the forms

$$*_{n,l}(q, y_1, \ldots, y_n), \quad \neg *_{n,l} (q, y_1, \ldots, y_n), \quad w \neq t.$$

Using tautologies $y_i = y_j \vee y_i \neq y_j$ and disjunctive splitting, we can assume that for each pair i, j either $y_i = y_j$ or $y_i \neq y_j$ occurs in C. If $y_i = y_j$ occurs in C, replace in C y_i by y_j. Assume that $z_1, \ldots, z_m$ is the pattern of $y_1, \ldots, y_n$ after such a replacement. Since for each pair i, j either $y_i = y_j$ or $y_i \neq y_j$ occurs in C, it is not hard to argue that the formula $\#^{z_1,\ldots,z_m}(z_1, \ldots, z_m)$ occurs in C. So we can assume that all B_i's have one of the forms

$$\#^{z_1,\ldots,z_m}(z_1, \ldots, z_m) \wedge *_{n,l}(w, y_1, \ldots, y_n),$$
$$\#^{z_1,\ldots,z_m}(z_1, \ldots, z_m) \wedge \neg *_{n,l} (w, y_1, \ldots, y_n),$$
$$w \neq t.$$

By Lemma 3.11 one can replace in these formulas $y_1, \ldots, y_n$ by the nonperiodic sequence $z_1, \ldots, z_m$, n by m, and l by a suitable k. So we can assume that all B_i's have one of the forms

$$\#^{z_1,\ldots,z_m}(z_1, \ldots, z_m) \wedge *_{m,k} (w, z_1, \ldots, z_m), \tag{19}$$
$$\#^{z_1,\ldots,z_m}(z_1, \ldots, z_m) \wedge \neg *_{m,k} (w, z_1, \ldots, z_m), \tag{20}$$
$$w \neq t, \tag{21}$$

where the sequence $z_1, \ldots, z_m$ is nonperiodic.

Concerning formulas of the form (21), note the following.

(22) Every formula $\exists w(w \neq t_1 \wedge \ldots \wedge w \neq t_v)$ is valid.

Indeed, every queue sort has an infinite number of elements, so we can always find a w different from $t_1, \ldots, t_v$.

If at least two different B_i's are of the form (19), then C contains the subformula

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1, \ldots, z_m) \wedge \\ *_{m,k}(w, z_1, \ldots, z_m) \wedge \\ \#^{u_1,\ldots,u_{m'}}(u_1, \ldots, u_{m'}) \wedge \\ *_{m',k'}(w, u_1, \ldots, u_{m'}). \end{array} \tag{23}$$

If $m \neq m'$, then by Lemma 3.13 this formula implies $length(w) < m + m'$, so we can eliminate the quantifier $\exists w$ as for formula (9) on page 6. If $m = m'$ but $k \neq k'$, then by Lemma 3.12 (23) is false, so we eliminate the quantifier again. If $m = m'$ and $k = k'$, then by Lemma 3.14 formula (23) implies

$$w = |z_1, \ldots, z_k| \vee z_1 = u_1 \wedge \ldots \wedge z_m = u_m.$$

Using disjunctive splitting, we can consider the cases $w = |z_1, \ldots, z_k|$ and $z_1 = u_1 \wedge \ldots \wedge z_m = u_m$ separately. If $w = |z_1, \ldots, z_k|$, we can again eliminate $\exists w$ as for formula (9) on page 6. If $z_1 = u_1 \wedge \ldots \wedge z_m = u_m$, then the (23) can be replaced by the equivalent formula

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1, \ldots, z_m) \wedge *_{m,k}(w, z_1, \ldots, z_m) \wedge \\ z_1 = u_1 \wedge \ldots \wedge z_m = u_m, \end{array}$$

which has less occurrences of subformulas of the form (19).

Thus, we can now assume that at least one of B_i's is either of the form (19) or of the form (20), but only at most one of B_i's is of the form (19). Consider two possible cases.

CASE: *all B_i's are of the form (20).* Then C is of the form

$$\exists w(\bigwedge_i (\#^{z_1,\ldots,z_{m_i}}(z_1, \ldots, z_{m_i}) \wedge \neg *_{m_i,k_i} (w, z_1, \ldots, z_{m_i})) \wedge w \neq t_1 \wedge \ldots \wedge w \neq t_v).$$

The variable w is of the sort $queue(\alpha)$ for some α. Note the following:

(24) *If the domain of α contains exactly one element, then $\neg *_{m,k} (w, z_1, \ldots, z_m)$ is false for all m, k.*

So for sorts α with one element C is equivalent to $\perp$. Assume that the domain of α has more than one element. By Lemma 3.10, if $\#^{z_1,\ldots,z_{m_i}}(z_1, \ldots, z_{m_i})$ holds only when w has period m_i. Let m be the maximum of all m_i's. It is not hard to argue that there is an infinite number of queues having no period between 1 and m. Each of these queues satisfies

$$\bigwedge_i \#^{z_1,\ldots,z_{m_i}}(z_1, \ldots, z_{m_i}) \wedge \neg *_{m_i,k_i} (w, z_1, \ldots, z_{m_i}),$$

since there is an infinite number of such queues, at least one of them also satisfies $w \neq t_1, \ldots, w \neq t_v$, so C can be replaced by $\top$.

CASE: *exactly one B_i is of the form (19).* We will now show how to get rid of all B_j's of the form (20). Suppose that at least one such B_j occurs in C, then C contains the subformula

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1,\ldots,z_m) \wedge *_{m,k}(w,z_1,\ldots,z_m) \wedge \\ \#^{u_1,\ldots,u_{m'}}(u_1,\ldots,u_{m'}) \wedge \\ \neg *_{m',k'}(w,u_1,\ldots,u_{m'}). \end{array} \tag{25}$$

Consider the following three possible cases.

SUBCASE: $m \neq m'$. Using the tautology

$$\mathit{length}(w) < m + m' \vee \neg(\mathit{length}(w) < m + m')$$

and disjunctive splitting, we can assume that either $\mathit{length}(w) < m+m'$ or $\neg(\mathit{length}(w) < m+m')$ occurs in C. In the former case, we can eliminate the quantifier as before. In the latter case, (25) becomes

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1,\ldots,z_m) \wedge \\ *_{m,k}(w,z_1,\ldots,z_m) \wedge \\ \#^{u_1,\ldots,u_{m'}}(u_1,\ldots,u_{m'}) \wedge \\ \neg *_{m',k'}(w,u_1,\ldots,u_{m'}) \wedge \\ \neg(\mathit{length}(w) < m + m'). \end{array} \tag{26}$$

By Lemma 3.13, the formula

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1,\ldots,z_m) \wedge *_{m,k}(w,z_1,\ldots,z_m) \wedge \\ \#^{u_1,\ldots,u_{m'}}(u_1,\ldots,u_{m'}) \wedge \\ \neg(\mathit{length}(w) < m + m') \end{array}$$

implies $\neg *_{m',k'}(w,u_1,\ldots,u_{m'})$, hence $\neg *_{m',k'}(w,u_1,\ldots,u_{m'})$ can be omitted from (26).

SUBCASE: $m = m'$ *and* $k \neq k'$. By Lemma 3.12

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1,\ldots,z_m) \wedge *_{m,k}(w,z_1,\ldots,z_m) \wedge \\ \#^{u_1,\ldots,u_{m'}}(u_1,\ldots,u_{m'}) \rightarrow \\ \qquad \neg *_{m',k'}(w,u_1,\ldots,u_{m'}), \end{array}$$

hence $\neg *_{m',k'}(w,u_1,\ldots,u_{m'})$ can be omitted from (25).

SUBCASE: $m = m'$ *and* $k = k'$. In this case (25) is

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1,\ldots,z_m) \wedge \\ *_{m,k}(w,z_1,\ldots,z_m) \wedge \\ \#^{u_1,\ldots,u_m}(u_1,\ldots,u_m) \wedge \\ \neg *_{m,k}(w,u_1,\ldots,u_m). \end{array} \tag{27}$$

Consider the tautology

$$\begin{array}{l} w = |z_1,\ldots,z_k| \vee (z_1 = u_1 \wedge \ldots \wedge z_m = u_m) \vee \\ \neg(w = |z_1,\ldots,z_k| \vee (z_1 = u_1 \wedge \ldots \wedge z_m = u_m)). \end{array}$$

Using this tautology and disjunctive splitting we can assume that one of the formulas $w = |z_1,\ldots,z_k|$, or $z_1 = u_1 \wedge \ldots \wedge z_m = u_m$, or $\neg(w = |z_1,\ldots,z_k| \vee (z_1 = u_1 \wedge \ldots \wedge z_m = u_m)$ occurs positively in C. If $w = |z_1,\ldots,z_k|$ occurs positively in C, then we can eliminate $\exists w$ as before. If $z_1 = u_1 \wedge \ldots \wedge z_m = u_m$ occurs positively in C, then (27) can be replaced by $\perp$. The only remaining case is that

$$\neg(w = |z_1,\ldots,z_k| \vee (z_1 = u_1 \wedge \ldots \wedge z_m = u_m))$$

occurs positively in C. By Lemma 3.14

$$\begin{array}{l} \#^{z_1,\ldots,z_m}(z_1,\ldots,z_m) \wedge *_{m,k}(w,z_1,\ldots,z_m) \wedge \\ \neg(w = |z_1,\ldots,z_k| \vee (z_1 = u_1 \wedge \ldots \wedge z_m = u_m)) \wedge \\ \#^{u_1,\ldots,u_m}(u_1,\ldots,u_m) \rightarrow \neg *_{m,k}(w,u_1,\ldots,u_m), \end{array}$$

hence $\neg *_{m,k}(w,u_1,\ldots,u_m)$ can be removed from (27).

In all three cases we could remove the subformula of the form (20). Therefore, we can assume that C has the form

$$\exists w(*_{m,k}(w,z_1,\ldots,z_m) \wedge w \neq t_1 \wedge \ldots \wedge w \neq t_v).$$

Using Lemma 3.10, one can see that there exists an infinite number of queues w satisfying $*_{m,k}(w,z_1,\ldots,z_m)$, so C is valid.

The proof of Lemma 3.7 is completed. □

LEMMA 3.15 *Let k be a nonnegative integer. The class of formulas $\exists xC$, where C is a conjunction of literals of the form (4), and the segment depth of x in C is $\leq k$, is a quantifier elimination reduction class for the class of such formulas but with the segment depth of x in C be $\leq k+1$.* □

Only a few final steps are to be made towards the proof of the Quantifier Elimination Theorem.

PROOF (of Theorem 3.1). By Lemma 3.7 it is enough to show how to eliminate the quantifier in $\exists x(A_1 \wedge \ldots \wedge A_n)$, where each A_i is of the form (4). By Lemma 3.15 we can decrease the segment depth of x to 0. When the segment depth is 0, the quantifier can be eliminated by Lemmas 3.6 and 3.7. □

Now let us rip the harvest.

THEOREM 3.16 *The first-order theory of any term algebra with queues is decidable.*

4 Axiomatization

Our proof gives us the following axiomatization of term algebras with queues. Such an axiomatization can be extracted from our proof, it is presented below. This axiomatization lacks elegance compared to the axiomatizations of term algebras. It is likely that our axiomatization can be slightly simplified, but we doubt that a considerable simplication is possible, since we cannot see how to avoid all combinatorics related to the formulas $*_{m,n}$. The axiomatization consists of two parts. One part is the standard axiomatization of term algebras, that can be found in [Hodges 1993] or [Maher 1988], including the following axioms:

$$x \neq t[x]$$

where x is a proper subterm of $t[x]$. In our case the term t may also contain operations on queues.

The second part consists of the following axioms

$$\begin{array}{l}
\varepsilon \neq \mathit{ladd}(x, w), \\
\varepsilon \neq \mathit{radd}(x, w), \\
w = \varepsilon \vee \exists x \exists w' (w = \mathit{ladd}(x, w')), \\
w = \varepsilon \vee \exists x \exists w' (w = \mathit{radd}(x, w')), \\
\mathit{ladd}(x_1, w_1) = \mathit{ladd}(x_2, w_2) \rightarrow x_1 = x_2 \wedge w_1 = w_2, \\
\mathit{radd}(x_1, w_1) = \mathit{radd}(x_2, w_2) \rightarrow x_1 = x_2 \wedge w_1 = w_2, \\
\mathit{ladd}(x, \mathit{radd}(y, w)) = \mathit{radd}(y, \mathit{ladd}(x, w)), \\
\mathit{ladd}(x_1, \ldots \mathit{ladd}(x_n, \varepsilon) \ldots) = \\
\qquad \mathit{radd}(x_n, \ldots \mathit{radd}(x_1, \varepsilon) \ldots), \\
|x_1 \ldots x_m \underline{w}| \neq |\underline{w} y_1 \ldots y_n|,
\end{array}$$

where in the last formula $m \neq n$, and sets of formulas (15), (16), (17), (18) used in our lemmas. Finally, for the very special case when a basic sort α has only one element, we have to add the axiom

$$\mathit{ladd}_\alpha(x, z) = \mathit{radd}_\alpha(x, z).$$

Completeness of this axiomatization can be checked by routine inspection of our proofs.

5 Related work and concluding remarks

[Bjørner 1998, Bjørner 2000] discusses the use of queues in verification and gives a decision procedure for the existential theory of queues, he uses a richer signature than we do. The signature of queues in [Bjørner 1998] contains the binary relation "subqueue", with such relations the first-order theory of queues (and even the $\forall\exists$-theory) is undecidable. Formulas expressing periodicity of queues also play a major role in [Bjørner 2000]. Enrichment of first-order theory of tree algebras with cyclic descriptions was also studied in [Peltier 1997].

[Maĺcev 1961] was the first to prove decidability of term algebras, he used quantifier elimination. Other quantifier elimination procedures are described in [Kunen 1987, Belegradek 1988*a*, Maher 1988, Comon and Lescanne 1989, Hodges 1993]. Each of [Maĺcev 1961, Maher 1988, Hodges 1993] gives an axiomatization of term algebras, Maher [1988] also provides such an axiomatization for rational trees. The technique of using decomposition functions f_i is due to [Kunen 1987, Belegradek 1988*a*].

The decision procedure of [Maĺcev 1961] gives a nonelementary upper bound on the first-order theory of term algebras. The first-order theory of term algebras also has a hereditary nonelementary lower bound. This result follows from [Compton and Henson 1990]: the theory of any paring function (i.e. binary one-to-one function) has a hereditary nonelementary lower bound. Obviously, such a function is definable in term algebras having at least one binary function symbol in the signature. When the signature contains only unary function symbols, the first-order theory of term algebras is LATIME($2^{O(n)}$): the class of problems solvable by alternating Turing machines running in exponential time but only with a linear number of alternations, for details see [Ferrante and Rackoff 1979, Berman 1980, Volger 1983*b*, Volger 1983*a*, Vorobyov and Voronkov 1998]. Vorobyov and Voronkov [1998] prove complexity results for some fragments of first-order logic over term algebras.

Evidently, if we consider the first-order theory of term algebras with queues, the nonelementary lower bound follows from the nonelementary lower bound for term algebras.

If we only consider queues over a finite domain, then our results proves the decidability of the first-order theory of words with the operations of the left and right multiplications by a letter. The weak monadic second-order theory of $0, 1$-words with right multiplication by 0 and 1 and left multiplication by 0 is known to be undecidable [Läuchly and Savoiz 1987, Seese 1972] [see also Thomas 1990].

If we only consider queues over a finite domain, then the lower bound LATIME($2^{O(n)}$) for queues follows from such a lower bound for term algebras with unary function symbols, since we can define term algebras with unary function symbols in any theory of queues, even queues over the domain of two elements. It is interesting to find the precise upper bound for the theory of queues over a finite (or infinite) domain, one can prove the upper bound LATIME($2^{O(n)}$) if the key result of [Ferrante and Rackoff 1979] can be proved for queues: quantifiers in any quantified formula can be bound by elements of the exponential height [see also Vorobyov and Voronkov 1998]. However, our proof does not imply such a bound on the number of elements in a queue. A related question is the complexity of any term algebra with queues and unary functions: one cannot directly represent a paring function in such structures, but these structures contain quite complex objects, for example queues of queues.

Acknowledgments

We thank Konstantin Korovin and Wolfgang Thomas.

References

BELEGRADEK O. [1988*a*], Model theory of locally free algebras (in Russian), *in* 'Model Theory and its Applications', Vol. 8 of *Trudy Instituta Matematiki*, Nauka, Novosibirsk, pp. 3–24. English summary in [Belegradek 1988*b*].

BELEGRADEK O. [1988*b*], Some model theory of locally free algebras (in Russian), *in* B. Dahn and H. Wolter, eds, '6th Easter Conference on Model Theory', Humboldt University, Berlin, pp. 28–32.

BERMAN L. [1980], 'The complexity of logical theories', *Theoretical Computer Science* **11**, 71–77.

BJØRNER N. [1998], Reactive verification with queues, *in* 'ARO/ONR/NSF/DARPA Workshop on Engineering Automation for Computer-Based Systems', Carmel, CA, pp. 1–8.

BJØRNER N. [2000], Integrating Decision Procedures for Temporal Verification, PhD thesis, Stanford University.

COMON H. AND LESCANNE P. [1989], 'Equational problems and disunification', *Journal of Symbolic Computations* **7**(3,4), 371–425.

COMPTON K. AND HENSON C. [1990], 'A uniform method for proving lower bounds on the computational complexity of logical theories', *Annals of Pure and Applied Logic* **48**, 1–79.

FERRANTE J. AND RACKOFF C. W. [1979], *The computational complexity of logical theories*, Vol. 718 of *Lecture Notes in Mathematics*, Springer-Verlag.

HODGES W. [1993], *Model theory*, Cambridge University Press.

KUNEN K. [1987], 'Negation in logic programming', *Journal of Logic Programming* **4**, 289–308.

LÄUCHLY H. AND SAVOIZ C. [1987], 'Monadic second-order definable relations on the binary tree', *Journal of Symbolic Logic* **52**, 219–226.

MAHER M. [1988], Complete axiomatizations of the algebras of finite, rational and infinite trees, *in* 'Proc. IEEE Conference on Logic in Computer Science (LICS)', pp. 348–357.

MAĹCEV A. [1961], 'On the elementary theories of locally free universal algebras', *Soviet Mathematical Doklady* **2**(3), 768–771. Reprinted in [Maĺcev 1971].

MAĹCEV A. [1971], Axiomatizable classes of locally free algebras of various types, *in* B. Wells III, ed., 'The Metamathematics of Algebraic Systems. Anatoliĭ Ivanovič Maĺcev. Collected Papers: 1936–1967', Vol. 66, North Holland, chapter 23, pp. 262–281.

PELTIER N. [1997], 'Increasing model building capabilities by constraint solving on terms with integer exponents', *Journal of Symbolic Computations* **24**(1), 59–101.

RYBINA T. AND VORONKOV A. [2000], A decision procedure for term algebras with queues, Preprint CSPP-2, Department of Computer Science, University of Manchester.
URL: *http://www.cs.man.ac.uk/preprints/index.html*

SEESE D. [1972], 'Entscheidbarkeits- und Definierbarkeitsfragen der Theorie "netzartiger" Graphen I', *Wiss. Zeitschr. der Humboldt-Univ. zu Berlin, Math.-Nat. Reihe* **21**, 513–517.

THOMAS W. [1990], Automata on infinite objects, *in* J. Van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, Elsevier Science, Amsterdam, chapter 4, pp. 133–191.

VISSER W., BARRINGER H., FELLOWS D., GOUGH G. AND WILLIAMS A. [1998], Efficient CTL model checking for analysis of Rainbow designs, *in* H. Li and K. Probst, eds, 'Advances in Hardware Design and Verification, Procedings of CHARME'97', Chapman and Hall, Montreal.

VOLGER H. [1983*a*], 'A new hierarchy of elementary recursive decision problems', *Methods of Operations Research* **45**, 509–519.

VOLGER H. [1983*b*], 'Turing machines with linear alternation, theories of bounded concatenation and the decision problem of first order theories (Note)', *Theoretical Computer Science* **23**, 333–337.

VOROBYOV S. AND VORONKOV A. [1998], Complexity of nonrecursive logic programs with complex values, *in* 'PODS'98', ACM Press, Seattle, Washington, pp. 244–253.